

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică

Departamentul Ingineria Software și Automatică

RAPORT

Structuri de date si algoritmi

A efectuat: st. gr.FAF-212 Lupascu Felicia

A verificat: dr. conf. Univ S. Corlat

**Problem 1A**

The constraint that the edges of each flagstone much be parallel to the edges of the square allows to analyze X and Y axes separately, that is, how many segments of length ‘a’ are needed to cover a segment of length ‘m’ and ’n’ — and take the product of these two quantities.

Answer = ceil(m/a) \* ceil(n/a), where ceil(x) is the least integer which is above or equal to x. Using integers only, it is usually written as ((m+a-1)/a)\*((n+a-1)/a). Note that answer may be as large as 10¹⁸, which does not fit in a 32-bit integer.

# include<bits/stdc++.h>  
using namespace std;  
int main()  
{  
 ios\_base:: sync\_with\_stdio(false);  
cin.tie(NULL);  
long long int n, m, a;  
cin >> n >> m >> a;  
cout << ((m + a - 1) / a) \* ((n + a - 1) / a);  
return 0;  
}



**Problem 2A**

This question lets you calculate the name with the largest suffix result.

The solution is to find the maximum value first, and then slowly increase it according to the initial sequence.

When the maximum value is found, it is very important to judge whether the final result is the maximum value. WA has been because of this several times.

#include<iostream>  
#include<cstring>  
#include<algorithm>  
#include<map>  
#include<string>  
using namespace std;  
map<string,int>q,every;  
struct node{  
 string name;  
 int k;  
}s[10005];  
int main()  
{  
 int n;  
 scanf("%d",&n);  
 int cnt = 0;  
 for(int i = 0;i <n; i ++)  
 {  
 cin>>s[i].name;  
 scanf("%d",&s[i].k);  
 q[s[i].name] += s[i].k;  
 }  
 map<string,int>::iterator it,its;  
 string ans = "";  
 int index = -1006;  
 for(it = q.begin(); it != q.end(); it ++)//Find the maximum  
 {  
 if(it->second> index)  
 {  
 index = it->second;  
 }  
 }  
 for(int i = 0; i <n; i ++)  
 {  
 //cout<<its->first<<" "<<its->second<<endl;  
 every[s[i].name] += s[i].k;  
  
 if(every[s[i].name]>=index&&q[s[i].name]>=index)  
 {  
 cout<<s[i].name;  
 break;  
 }  
 }  
}



**Problem 3A**

The gist of the title is, given a king's starting point and ending point, find the shortest path from the starting point to the ending point for the king.

(The king can only move to one of the 8 surrounding squares at a time)

This is simple greed.

1. Find the displacement dx in the x direction

2. Find the displacement dy in the y direction

3. Go min(abs(dx), abs(dy)) first

4. Walk max(abs(dx), abs(dy)) - min(abs(dx), abs(dy)) straight

#include <iostream>  
#include <cmath>  
using namespace std;  
int main()  
{  
 char s1[5],s2[5];  
 while(cin>>s1>>s2)  
 {  
 int x=s2[0]-s1[0];//LR  
 int y=s2[1]-s1[1];//UD  
 cout<<max(abs(x),abs(y))<<endl;  
 while(x||y)  
 {  
 if(x>0)  
 x--,cout<<"R";  
 if(x<0)  
 x++,cout<<"L";  
 if(y>0)  
 y--,cout<<"U";  
 if(y<0)  
 y++,cout<<"D";  
 cout<<endl;  
 }  
 }  
 return 0;  
}



**Problem 4A**

The explanation here is:

* No odd number can be divided into two even parts --> odd-even = odd
* Almost every even number can be divided into two even parts --> even - even = even
* Edge case is 2; as it can NOT be divided into two even parts, 1 is an odd number, and can not be divided into 2 & 0 as one boy will not take part in addition to that 0 is not an even number

#include<bits/stdc++.h>   
using namespace std;   
int main() {  
 int t;  
 cin>>t;  
 if(t==2)  
 cout<<"NO"<<endl;  
 else if(t%2==0)  
 cout<<"YES"<<endl;  
 else  
 cout<<"NO"<<endl;  
return 0; }



**Problem 5A**

In a chat server, "+"+person's name indicates that the person joins the chat, "-"+person's name indicates that the person should exit the chat, and the content after the colon after the person's name indicates the message sent by the person, and the message sent by a person can be used by anyone who is currently chatting. Received, ask how many characters are received in total.

Simple simulation, note: the message sent is after the colon

#include <iostream>  
#include <cstring>  
#include <cstdio>  
using namespace std;  
int main()  
{  
 char a[100];  
 int sum=0;  
 int ans=0;  
 while(gets(a))  
 {  
 int len=strlen(a);  
 if(a[0]=='+')  
 sum++;  
 else if(a[0]=='-')  
 sum--;  
 else  
 {  
 int p=0;  
 for(int i=0;i<len;i++)  
 if(a[i]==':')  
 p=i;  
 ans+=(len-p-1)\*sum;  
 }  
 }  
 cout<<ans<<endl;  
 return 0;  
}



**Problem 6A**

Give 4 sides and ask whether it can form an undegenerate triangle. A degenerate triangle cannot form any triangle

Theorem for judging triangles: the sum of the two smaller sides is greater than the third side.

If they are equal, the triangle degenerates into a straight line.

Check if it's possible to form a triangle, then check if it's possible to form a segment.

#include<iostream>  
#include<algorithm>  
using namespace std;  
int main(){  
 int arr[4];  
 for(int i=0;i<4;i++)  
 cin>>arr[i];  
 sort(arr,arr+4);  
 //Non-degenerate triangle  
 if(arr[0]+arr[1]>arr[2]||arr[0]+arr[1]>arr[3]||arr[1]+arr[2]>arr[3]|| arr[0]+arr[2]>arr[3]){  
 cout<<"TRIANGLE"<<endl;  
 return 0;  
 }  
 //Degenerate triangle  
 if(arr[0]+arr[1]==arr[2]||arr[0]+arr[1]==arr[3]||arr[1]+arr[2]==arr[3 ]||arr[0]+arr[2]==arr[3]){  
 cout<<"SEGMENT"<<endl;  
 return 0;  
 }  
 //Cannot form any triangle  
 cout<<"IMPOSSIBLE"<<endl;  
 return 0;  
}



**Problem 7A**

8\*8 disc initializes the color of each point to white. You can now paint the tablet, but you can only paint one row or column at a time. Now looking at the coated tablet, you need to apply it at least several times.

Ideas: Count the first row and column directly to judge violently, but it should be noted that if the entire tablet is black only 8 times

In the beginning, the chessboard is all white. Input the chessboard required by the customer, and output at least a few strokes to meet the customer's request.

Determine whether each row/column is all B, and if it is all B, increase the number of times of brushing.

Note that if the number of times brushing is 16, it means that all of them are B. At this time, only 8 times are needed.

#include <cstdio>  
#include <iostream>  
using namespace std;  
char m[10][10];  
int main(void)  
{  
 int cnt, ans = 0;  
 for (int i = 0; i < 8; i++)  
 cin >> m[i];  
 for (int i = 0; i < 8; i++)  
 {  
 cnt = 0;  
 for (int j = 0; j < 8; j++)  
 {  
 if (m[i][j] == 'B')  
 cnt++;  
 }  
 if (cnt == 8)  
 ans++;  
 }  
 for (int j = 0; j < 8; j++)  
 {  
 cnt = 0;  
 for (int i = 0; i < 8; i++)  
 {  
 if (m[i][j] == 'B')  
 cnt++;  
 }  
 if (cnt == 8)  
 ans++;  
 }  
 if (ans == 16)  
 ans = 8;  
 cout << ans << endl;  
 return 0;  
}



**Problem 8A**

The meaning of the question:give you three strings s1, s2, s3.

Ask whether there are two substrings of s2 and s3 in s1, and it is required that s2 and s3 do not intersect and s2 is in front of s3.

# include <bits/stdc++.h>  
# define ll long long int  
# define pi 3.141592653  
# define pb push\_back  
# define mp make\_pair  
# define all(a) a.begin(), a.end()  
# define F first  
# define S second  
using namespace std;  
// Multiplies str1 and str2, and prints result.  
string multiply(string num1, string num2)  
{  
 int len1 = num1.size();  
int len2 = num2.size();  
if (len1 == 0 | | len2 == 0)  
return "0";  
// will keep the resulting number in vector in reverse order  
vector < int > result(len1 + len2, 0);  
// Below two indexes are used to find positions in the result.  
int i\_n1 = 0;

Int i\_n2 = 0;  
// Go from right to left in num1  
for (int i=len1-1; i >= 0; i--)  
 {  
 int carry = 0;  
 int n1 = num1[i] - '0';  
 // Go from right to left in num2  
 for (int j=len2-1; j >= 0; j--)  
 {  
 // Take the current digit of the second number  
 int n2 = num2[j] - '0';  
 // Multiply with a current digit of the first number and add result to previously-stored result at the current position.  
 int sum = n1 \* n2 + result[i\_n1 + i\_n2] + carry;  
 // Carry for next iteration  
 carry = sum / 10;  
 // Store result  
 result[i\_n1 + i\_n2] = sum % 10;  
 i\_n2++;  
 }  
 // store carry in next cell  
 if (carry > 0)  
 result[i\_n1 + i\_n2] += carry;  
 // To shift pos. to left after every multiplication.of a digit in num1.  
 i\_n1++;  
 }   
// ignore '0' s from the right  
int i = result.size() - 1;  
while (i >= 0 & & result[i] == 0)  
 i - -;  
// If all were '0's – means either both or one of num1 or num2 were '0'  
if (i == -1)  
 return "0";  
// generate the result string  
string s = "";  
while (i >= 0)  
 s += std::to\_string(result[i - -]);  
return s;  
}  
int isPrime(ll n)  
{  
// Corner case  
if (n <= 1)  
 return 0;  
// Check from 2to n - 1  
for (ll i = 2; i < n; i++)  
 if (n % i == 0)  
 return 0;  
return 1;}  
ll modFact(ll n, ll p)  
{  
if (n >= p)  
 return 0;  
ll result = 1;  
for (ll i = 1; i <= n; i++)  
 result = (result \* i) % p;  
return result;}  
int main()  
{  
ios\_base::sync\_with\_stdio(false);  
cin.tie(NULL);  
string s, s1, s2;  
cin >> s >> s1 >> s2;  
ll  
x = s.length();  
ll y = s1.length();  
ll z = s2.length();  
ll f1 = 0;  
ll f2 = 0;  
ll ind = -1;  
for (ll i=0;i < x;i++)  
 {  
 ll k = i;  
 ll k1 = 0;  
while (k1 < y & & k < x)  
 {  
 if (s1[k1] != s[k])  
 {  
 break;  
 }  
 else  
 {  
 k1 + +;  
 k + +;  
}}  
if (k1 == y)  
{  
ind=k;  
break;  
}}  
if (ind != -1)  
{  
for (ll i=ind;i < x;i++)  
{  
ll k = i;  
ll k1 = 0;  
while (k1 < z & & k < x)  
 {  
 if (s2[k1] != s[k])  
 {  
break;  
}  
else  
{  
k1 + +;  
k + +;  
}}  
if (k1 == z)  
{  
f1=1;  
break;

}}}  
ind = -1;  
for (ll i=x-1;i >= 0;i--)  
{  
 ll k = i;  
 ll k1 = 0;  
while (k1 < y & & k >= 0)  
 {  
 if (s1[k1] != s[k])  
 {  
break;  
}  
else  
{  
k1 + +;  
k - -;  
}}  
if (k1 == y)  
{  
ind=k;  
break;  
}}  
// cout << ind;  
if (ind != -1)  
{  
for (ll i=ind;i >= 0;i--)  
{  
 ll k = i;  
ll k1 = 0;  
while (k1 < z & & k >= 0)  
 {  
 if (s2[k1] != s[k])  
 {  
break;}  
else  
{  
k1 + +;  
k - -;  
}}  
if (k1 == z)  
{  
f2=1;  
break;}  
}}  
// cout << f1 << " " << f2;  
if (f1 == 1 & & f2 == 1)  
cout << "both";  
else if (f1 == 1)  
cout << "forward";  
else if (f2 == 1)  
cout << "backward";  
else  
cout << "fantasy";  
return 0;  
}



**Problem 9A**

#include <iostream>  
#include <string>  
using namespace std;  
int main()  
{  
 int Y, W;  
 cin >> Y >> W;  
 const string probability[7] = {"", "1/1", "5/6", "2/3", "1/2", "1/3", "1/6"};  
 int D = max(Y, W);  
 cout << probability[D] << endl;  
 return 0;  
}  
}



**Problem 10A**

// n is the number of working intervals

// P1, P2, P3 is the power consumption

// T1 is the non-moving time to reach P1 --> P2

// T2 is the non-moving time to reach P2 --> P3

// Total is the total power consumption

#include<stdio.h>  
int p1, p2, p3;  
int t1, t2;  
int n;  
int pp = 0, p, k;  
int i;  
int ret;  
int main()  
{  
 scanf("%d%d%d%d%d%d", &n, &p1, &p2, &p3, &t1, &t2);  
 scanf("%d%d", &p, &k);  
  
 ret = (k-p)\*p1; //+ p3\*p;  
 for(i = 1; i < n; i++)  
 {  
 pp = k;  
 scanf("%d%d", &p, &k);  
 if(p - pp > t1)  
 {  
 ret += t1\*p1;  
 pp += t1;  
 if(p - pp > t2)  
 ret += t2\*p2 + (p - pp - t2)\*p3;  
 else  
 ret += (p-pp)\*p2;  
 }  
 else  
 ret += (p-pp)\*p1;  
 ret += (k-p)\*p1;  
 }  
 printf("%d", ret);  
 getchar();  
 getchar();  
 return 0;  
}



**Problem 11A**

#include <iostream>  
using namespace std;  
int main()  
{  
int n,d,a[2005],ans = 0;  
cin>>n>>d;  
for(int i = 1;i <= n;i++) cin>>a[i];  
for(int i = 2;i <= n;i++)//Start from 2  
{  
if(a[i] <= a[i - 1])

//This number is compared with the previous number  
 {  
 ans += (a[i - 1] - a[i]) / d + 1;

//Using the formula  
 a[i] += ((a[i - 1] - a[i]) / d + 1) \* d;

//update number  
 }  
}  
cout<<ans<<endl;  
return 0;  
}



**Problem 12A**

#include <bits/stdc++.h>  
using namespace std;  
typedef long long int ll;  
typedef long double ld;  
typedef vector<int> vi;  
typedef multiset<int> ms;  
typedef set<int> s;  
typedef vector<double> vid;  
typedef multiset<double> msd;  
typedef set<double> sd;  
#define sp setprecision  
#define N 100005  
#define MOD 1000000007  
#define dd double  
#define rep(i, n) for(int i = 0; i < n; i++)  
#define REP(i,a,b) for(int i=a;i<b;i++)  
#define rep1(i,b) for(int i=1;i<=b;i++)  
#define pb push\_back  
#define mp make\_pair  
#define clr(x) x.clear()  
#define sz(x) ((int)(x).size())  
#define F first  
#define S second  
void fastscan(ll &number)  
{  
 //variable to indicate sign of input number  
 bool negative = false;  
 register int c;  
 number = 0;  
 // extract current character from buffer  
 c = getchar();  
 if (c=='-')  
 {  
 // number is negative  
 negative = true;  
 // extract the next character from the buffer  
 c = getchar();  
 }  
 // Keep on extracting characters if they are integers  
 // i.e ASCII Value lies from '0'(48) to '9' (57)  
 for (; (c>47 && c<58); c=getchar())  
 number = number \*10 + c - 48;  
 // if scanned input has a negative sign, negate the  
 // value of the input number  
 if (negative)  
 number \*= -1;  
}  
// Function Call  
int main()  
{  
 ios\_base::sync\_with\_stdio(false);  
 cin.tie(NULL);  
 //ios\_base& scientific (ios\_base& str);  
 char a,s,d,f,g,h,j,k,l;  
 cin>>a>>s>>d>>f>>g>>h>>j>>k>>l;  
 if(a==l && s==k && d==j && f==h) cout<<"YES"; else cout<<"NO";  
 return 0;  
}



**Problem 13A**

First, analyze how to calculate the sum of each digit of each hexadecimal number. The first thing that comes to mind is array storage and then summation. Each bit stored can be obtained by taking the remainder. This is the basic idea.

However, we must also consider the problem of the simplest fraction. To simplify the fraction, it is essential to find the greatest common divisor of two numbers. You can use gcd recursion, and then you can use the division method.

#include<iostream>  
#include<string>  
#include<cstring>  
#include<algorithm>  
#include<cmath>  
#include<vector>  
using namespace std;  
int sum = 0;  
int gcd(int x,int y)  
{  
 if(!y)  
 {  
 return x;  
 }  
 else  
 {  
 return gcd(y,x%y);  
 }  
}  
void trans(int n,int x)  
{  
 int q = 0;  
 int a[1000] = {  
 0};  
 while(n)  
 {  
 a[q++] = n%x;  
 n /= x;  
 }  
 for(int i = 0;i < q;i++)  
 {  
 sum += a[i];  
 }  
}  
int main()  
{  
 int n;  
 cin>>n;  
 for(int i = 2;i <= n-1;i++)  
 {  
 trans(n,i);  
 }  
 cout<<sum/gcd(sum,n - 2)<<"/"<<(n - 2)/gcd(sum,n - 2)<<endl;  
 return 0;  
}



**Problem 14A**

To find the smallest rectangle containing the picture, iterate through the pairs (i,j) such that the j-th symbol in i-th line is '\*'; find the minimum and maximum values of i and j from these pairs. The rectangle to output is [imin, imax] × [jmin, jmax].

#include<map>  
#include<list>  
#include<cmath>  
#include<cstdio>  
#include<cstring>  
#include<iostream>  
#include<algorithm>  
using namespace std;  
int n,m,p,q,x,y;  
char gg,a[101][101];  
int main(){  
scanf("%d%d",&n,&m);  
p=0x3f,x=0x3f;//infinity  
for(int i=1;i<=n;i++)  
for(int j=1;j<=m;j++){  
cin>>gg;  
a[i][j]=gg;  
if(gg=='\*'){  
if(i<=p) p=i**;//The top left corner\***if(j<=x) x=j;//The top right corner\*  
if(i>=q) q=i;//The bottom left corner  
if(j>=y) y=j;//The bottom right corner  
}  
}  
for(int i=p;i<=q;i++){  
for(int j=x;j<=y;j++)  
printf("%c",a[i][j]);  
putchar('\n');  
}  
return ~~(0-0);  
}



**Problem 15A**

It is possible for the x-coordinate of the new house to have a non-integer value.

This question needs to be noted that where the length of the house if you want, then divided by 2, you need to save to floating-point type.

#include <cstdio>  
#include <cstring>  
#include <iostream>  
#include <algorithm>  
using namespace std;  
typedef pair<double, double> p;  
p arr[1005];  
bool cmp(p a, p b)  
{  
 return a.first < b.first;  
}  
int main(void)  
{  
 int n, t, ans = 2;  
 cin >> n >> t;  
 for (int i = 0; i < n; i++)  
 cin >> arr[i].first >> arr[i].second;  
 sort(arr, arr + n, cmp);  
 for (int i = 0; i < n - 1; i++)  
 {  
 if (arr[i].first + arr[i].second / 2 + arr[i + 1].second / 2 + t < arr[i + 1].first)  
 ans += 2;  
 else if (arr[i].first + arr[i].second / 2 + arr[i + 1].second / 2 + t == arr[i + 1].first)  
 ans++;  
 }  
 cout << ans << endl;   
 return 0;  
}



**Problem 16A**

The meaning of the question: If the numbers in each line are the same, but the adjacent lines are different, YES is output.

Two adjacent lines are different characters

Every line is the same character

#include <iostream>  
#include <cstring>  
#include <cstdlib>  
#include <stdio.h>  
#include <algorithm>  
#include <vector>  
#include <queue>  
#include <set>  
#include <map>  
#include <string>  
#include <math.h>  
#include <stdlib.h>  
#include <iomanip>  
#include <list>  
#include <deque>  
#include <stack>  
#define ull unsigned long long  
#define ll long long  
#define mod 90001  
#define INF 1<<30  
#define maxn 10000+10  
#define cle(a) memset(a,0,sizeof(a))  
const ull inf = 1LL << 61;  
const double eps=1e-5;  
using namespace std;  
bool cmp(int a,int b){  
return a>b;  
}  
string s;  
int main()  
{  
#ifndef ONLINE\_JUDGE  
//freopen("in.txt","r",stdin);  
#endif  
//freopen("out.txt","w",stdout);  
int n,m;  
while(cin>>n>>m){  
s="";  
string t="";  
int mark=0;  
for(int i=1;i<=n;i++){  
if(i==1){  
cin>>s;  
for(int i=1;i<m;i++)  
if(s[i]!=s[i-1])mark=1;  
}  
else {  
cin>>t;  
if(t==s){  
mark=1;  
}  
if(t!=s){  
for(int i=1;i<m;i++)  
if(t[i]!=t[i-1])mark=1;  
}  
s=t;  
}  
}  
if(mark)cout<<"NO";  
else cout<<"YES";  
}  
return 0;  
}



**Problem 17A**

In the first sample the answer is YES since at least two numbers can be expressed as it was described (for example, 13 and 19). In the second sample the answer is NO since it is impossible to express 7 prime numbers from 2 to 45 in the desired form.

a boolean vector of length 1010 such that B[i] is true, if it is Noldbach number and false otherwise.

Now, instead of iterating from i = 1 to N in the given range to check the given conditions ...Pick up the primes in the precomputed vector and check if they're Noldbach numbers ... This means we have to do log N iterations instead of O(N) for each query ...

However precomputation is O(N log log N + log N).

# include<bits/stdc++.h>  
using namespace std;  
int n, m, k, r = 0, j, b[1010];  
bool a[1010];  
int main()  
{  
 cin >> n >> k;  
for (int i=2;i <= n;i++)  
{  
if (!a[i])  
{  
 b[m + +] = i;  
for (j=i \* 2;j <= n;j += i)  
a[j] = true;  
}  
}  
for (int i=0;i < m;i++){  
for (j=0;j < i;j++) if (b[i]-1-b[j] == b[j+1]){  
r++;  
break;  
}  
}  
if (r >= k) cout << "yes";  
else cout << "no";  
return 0;



**Problem 18A**

This question is asking, there are three points on the lattice matrix, can they form a right triangle, if they can, it is Right, but now if one of the points moves one square, it becomes Right, then it is ALMOST (almost), Otherwise, it is NEITHER. In fact, it is simple. If not, enumerate the four points around the three points, a total of 12 points, enumerate, answer, and finish.

Regarding the judgment, the sum of the squares of the two sides of the triangle is still equal to the sum of the squares of the third side, and if the side is not horizontal or vertical, use the horizontal and vertical sides of the side projection instead of the square sum ~ is also the Pythagorean theorem~~~

# include<iostream>   
# include<algorithm>  
using namespace std;  
# define Jx(a) ((i==a)?tx:x1[a])  
# define Jy(a) ((i==a)?ty:y1[a])  
int m[4][2] = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};  
bool isR(int x1, int y1, int x2, int y2, int x3, int y3)  
{  
 int a, b, c;  
a = (x1 - x2) \* (x1 - x2) + (y1 - y2) \* (y1 - y2);  
b = (x1 - x3) \* (x1 - x3) + (y1 - y3) \* (y1 - y3);  
c = (x3 - x2) \* (x3 - x2) + (y3 - y2) \* (y3 - y2);  
if (a == 0 | | b == 0 | | c == 0)  
return 0;  
if (a + b == c | | a + c == b | | b + c == a) return 1;  
else return 0;  
}  
int main()  
{  
int x1[4], y1[4];  
for (int i=0;i < 3;i++) cin >> x1[i] >> y1[i];  
if (isR(x1[0], y1[0], x1[1], y1[1], x1[2], y1[2]))  
 {  
 cout << "RIGHT" << endl;  
return 0;  
}  
int tx, ty;  
for (int i=0;i <= 3;i++)  
{  
for (int j=0;j <= 3;j++)  
 {  
 tx = x1[i] + m[j][0];  
 ty = y1[i] + m[j][1];  
 if (isR(Jx(0), Jy(0), Jx(1), Jy(1), Jx(2), Jy(2)))  
 {  
 cout << "ALMOST" << endl;  
return 0;  
}  
  
/ \*\* \*\* \*The Complete Version Below \*\* \*\* \*  
if (i == 0 & & isR(tx, ty, x1[1], y1[1], x1[2], y1[2]))  
{  
cout << "ALMOST" << endl;  
return 0;  
}  
if (i == 1 & & isR(x1[0], y1[0], tx, ty, x1[2], y1[2]))  
{  
cout << "ALMOST" << endl;  
return 0;  
}  
if (i == 2 & & isR(x1[0], y1[0], x1[1], y1[1], tx, ty))  
{  
cout << "ALMOST" << endl;  
return 0;  
}  
\*\* \*\* \*The Complete Version Above \*\* \*\* \* /  
}  
if (i == 3) cout << "NEITHER" << endl;  
}  
return 0;  
}



**Problem 19A**

Simulate a football match and calculate the qualifying situation. Just sort them in two order.

#include <bits/stdc++.h>  
using namespace std;  
const int N = 55;  
struct C {  
 string na;  
 int sc, jq, sq;  
} a[N];  
unordered\_map<string, int> p;  
int n, nu1, nu2;  
string na1, na2, s;  
  
bool cmp(C a, C b) {  
 if (a.sc != b.sc) {  
 return a.sc > b.sc;  
 } else if (a.jq - a.sq != b.jq - b.sq) {  
 return a.jq - a.sq > b.jq - b.sq;  
 } else {  
 return a.jq > b.jq;  
 }  
}  
bool cmp2(C a, C b) {  
 return a.na < b.na;  
}  
  
int main() {  
 cin >> n;  
 for (int i = 1; i <= n; ++i) {  
 cin >> a[i].na;  
 p[a[i].na] = i;  
 }  
 for (int i = 1; i <= n \* (n - 1) / 2; ++i) {  
 cin >> s;  
 for (int j = 0; j < s.size(); ++j) {  
 if (s[j] == '-') {  
 na1 = s.substr(0, j);  
 na2 = s.substr(j + 1, s.size() - j - 1);  
 break;  
 }  
 }  
 cin >> s;  
 nu1 = nu2 = 0;  
 bool f = 0;  
 for (int j = 0; j < s.size(); ++j) {  
 if (s[j] == ':') {  
 f = 1;  
 } else if (f) {  
 nu2 = nu2 \* 10 + s[j] - '0';  
 } else {  
 nu1 = nu1 \* 10 + s[j] - '0';  
 }  
 }  
 a[p[na1]].jq += nu1;  
 a[p[na1]].sq += nu2;  
 a[p[na2]].jq += nu2;  
 a[p[na2]].sq += nu1;  
 if (nu1 > nu2) {  
 a[p[na1]].sc += 3;  
 } else if (nu2 > nu1) {  
 a[p[na2]].sc += 3;  
 } else {  
 a[p[na1]].sc += 1;  
 a[p[na2]].sc += 1;  
 }  
 }  
 sort(a + 1, a + n + 1, cmp);  
 sort(a + 1, a + n / 2 + 1, cmp2);  
 for (int i = 1; i <= n / 2; ++i) {  
 cout << a[i].na << '\n';  
 }  
 return 0;  
}



**Problem 20A**

The meaning is almost to /delete the repeated examples in the string given to you. It is very intuitive.

The replacement is to replaced check the cppreference and finds that there is such a std::string::replacestring.replace(char, char), but it is not as simple as C#. This std::string::replace(int pos, int count, string str)is to replace the characters string from the post beginning with count str

# include <bits/stdc++.h>  
using namespace std;  
int main()  
{  
 string a;  
cin \>> a;  
L1:  
if (a.find("//" ,0) == string::npos)  
goto  
L2;  
a = a.replace(a.find("//",0),2,"/");  
goto L1;  
L2:  
 if (a[a.length() - 1] =='/' & & a.length() !=1)  
a = a.substr(0, a.length() -1);  
cout \<< a;  
return 0;  
}



**Problem 12B**

#include<bits/stdc++.h>  
using namespace std;  
int main()  
{  
 string s,t;  
 cin>>s>>t;  
 sort(s.begin(),s.end()); s[s.length()]='0';  
 if(s[0]=='0') swap(s[0],s[1]);  
 if(t==s) cout<<"OK";  
 else cout<<"WRONG\_ANSWER";  
}

****

**Problem 63A**

This problem can be easily solved with the following algorithm. First, read the names and statuses of all crew members into an array. Next, iterate over the entire array 4 times. In the first pass output names of rats. In the second pass output names of women and children. In the third pass output names of men. Finally, in the fourth pass output the name of the captain.

# include <bits/stdc++.h>  
using namespace std;  
int main()  
{  
 int n, i;  
string name[105], status[105];  
cin >> n;  
for (i = 0; i < n; i++)  
{  
 cin >> name[i] >> status[i];  
}  
for (i = 0; i < n; i++) if (status[i] == "rat") cout << name[i] << endl;  
for (i = 0; i < n; i++) if (status[i] == "woman" | | status[i] == "child") cout << name[i] << endl;  
for (i = 0; i < n; i++) if (status[i] == "man") cout << name[i] << endl;  
for (i = 0; i < n; i++) if (status[i] == "captain") cout << name[i] << endl;  
return 0;  
}



**Problem 600B**

Let's sort all numbers in a. Now let's iterate over elements of *b* and for element *bj* find the index of the lowest number that is greater than *bj*. We can do that using binary search. That index will be the answer for value *bj*.

Complexity: *O*(*nlogn*).

#include<bits/stdc++.h>  
using namespace std;  
typedef long long int ll;  
int main()  
{  
 ll a,b;  
 ll i,j,k;  
 cin>>a>>b;  
 ll x[a],y[b];  
 for(i=0;i<a;i++)  
 cin>>x[i];  
 for(i=0;i<b;i++)  
 cin>>y[i];  
 sort(x,x+a);  
 for(i=0;i<b;i++)  
 {  
 cout<<upper\_bound(x,x+a,y[i])-x<<" ";  
 }  
 return 0;  
}



**Problem 1450A**

The string "trygub" is not sorted alphabetically, and a subsequence of a sorted string is necessarily sorted. So, if we sort the input string, it will be a solution. Complexity is O(n) with counting sort.

#include <stdio.h>  
int main(){  
 int t;  
 char s[201];  
 scanf("%d",&t);  
 while(t--){  
 int size;  
 scanf("%d",&size);  
 scanf("%s",s);  
 for(int i=0;i<size;i++){  
 if(s[i]=='b'){  
 for(int j=i;j>0;j--){  
 s[j]=s[j-1];  
 }  
 s[0]='b';  
 }  
 }  
 printf("%s\n",s);  
 }  
 return 0;  
}



**Problem 8B**

From any position, move up, down, right and left according to UDRL as a bearing guide (draw a route on paper)

The question asks if the route from the starting point (your own starting point) to the ending point (the position after the last move) is the shortest (that is, don't go around in circles)

#include<iostream>  
#include<cstring>  
using namespace std;  
int sx=200,sy=200;  
int dir[4][2]={{0,1},{0,-1},{-1,0},{1,0}};  
int vis[550][550];  
string a;  
int main()  
{  
//1. The road is not repeated, that is, the next point reached cannot be visited  
//2. Only this time has been reached around the point where I walked, and the other three places have not been visited.  
vis[sx][sx]=1; //Initial position  
cin>>a;  
for(int i=0;i<a.length();i++)  
{  
if(a[i]=='U')  
sx--;  
else if(a[i]=='D')  
sx++;  
else if(a[i]=='L')  
sy--;  
else sy++;  
if(vis[sx][sy]) //If the next point we reach has been walked, output the bug directly  
{  
cout<<"BUG";  
return 0;  
}  
vis[sx][sy]=1; //Otherwise the next point passed is marked as 1  
int count=0;  
for(int j=0;j<4;j++)  
{  
if(vis[sx+dir[j][0]][sy+dir[j][1]]) //Find a way in four directions  
count++;  
}  
if(count!=1)  
{  
cout<<"BUG";  
return 0;  
}  
}  
cout<<"OK";  
}



**Problem 217A**

Notice that the existence of a snowdrift at the point (*x*, *y*) implies that "if I'm on the horizontal line at *y* then I am certainly able to get to the vertical line at *x*, and vice versa". Thus, the snowdrifts are the edges of a bipartite graph between x- and y- coordinates. The number of snowdrifts that needs to be added to make this (as well as the original) graph connected is the number of its connected components reduced by one.

# include <bits/stdc++.h>  
using namespace std;  
int n, realCmp, vis[111];  
struct point  
{  
 int x, y;  
point()  
{  
 x = y = 0;  
}  
};  
point  
points[111];  
vector < vector < int > > g;  
void  
dfs(int  
u){  
 vis[u] = 1;  
for (int i=0;i < g[u].size();  
i + +)  
if (!vis[g[u][i]])  
dfs(g[u][i]);  
}  
int  
main()  
{  
 cin >> n;  
for (int i=0;i < n;i++)  
cin >> points[i].x >> points[i].y;  
g.resize(n);  
for (int i=0;i < n;i++)  
for (int j=i+1;j < n;j++)  
if (points[i].x == points[j].x | | points[i].y == points[j].y)  
g[i].push\_back(j), g[j].push\_back(i);  
for (int i=0;i < n;i++)  
if (!vis[i])  
dfs(i), realCmp++;  
cout << realCmp-1;  
return 0;  
}



**Problem 500A**

In this problem we are given a directed graph, and asked whether a particular vertex is reachable from vertex 1. It is possible to solve this by running a depth-first search starting from vertex 1.

Since every vertex has at most one outgoing edge, it is possible to write the DFS as a simple loop

#include <bits/stdc++.h>  
using namespace std;  
int a[30001];  
int main(){  
 int i,n,t;  
 cin>>n>>t;  
 for(i=1;i<=n;i++){  
 cin>>a[i];  
 }  
 int x=1;  
 while(x<t){  
 x=x+a[x];  
 }  
 if(x==t){  
 cout<<"YES";  
 return 0;  
 }  
 cout<<"NO";  
}



**Problem 1487A**

If for some hero i, no other hero is weaker than i, then the i-th hero cannot win any fights and is not a possible winner. Otherwise, the hero i is a possible winner — he may fight the weakest hero 100^500 times and be declared the winner.

So the solution to the problem is calculating the number of minimum elements in the array a, since all other elements denote possible winners of the tournament.

#include <bits/stdc++.h>  
using namespace std;  
int main() {  
 int t;  
 cin >> t;  
 while (t--) {  
 int n;  
 cin >> n;  
 vector<int> a(n);  
 for (int& x : a) cin >> x;  
 cout << n - count(a.begin(), a.end(), \*min\_element(a.begin(), a.end())) << endl;  
 }  
}



**Problem 245H**

Strings and arrays are considered 0-based in the following solution. Let isPal[i][j] be 1 if s[i...j] is palindrome, otherwise, set it 0. Let's define dp[i][j] to be number of palindrome substrings of s[i...j]. Let's calculate isPal[i][j] and dp[i][j] in O(|S|2). First, initialize isPal[i][i] = 1 and dp[i][i] = 1. After that, loop over len which states length of substring and for each specific len, loop over start which states starting position of substring. isPal[start][start + len - 1] can be easily calculated by the following formula:

isPal[start][start+len-1] = isPal[start+1][start+len-2] & (s[start] == s[start+len-1])

After that, dp[start][start + len - 1] can be calculated by the following formula which is derived from [Inc-Exc Principle](http://en.wikipedia.org/wiki/Inclusion-exclusion_principle).

dp[start][start+len-1] = dp[start][start+len-2] + dp[start+1][start+len-1] - dp[start+1][start+len-2] + isPal[start][start+len-1]

After preprocessing, we get queries li and ri and output dp[li - 1][ri - 1]. Overall complexity is O(|S|2).

#include<bits/stdc++.h>  
using namespace std;  
bool is\_pal[5001][5001];  
int n,arr[5001][5001];  
char s[5001];  
int main() {  
 scanf("%s%d", &s, &n);  
 int l=strlen(s);  
 for(l=0;s[l];l++){};  
 for (int i=l-1;~i;i--){  
 for (int j=i;j<l;j++) {  
 is\_pal[i][j] = s[i]==s[j] && (i+1>=j || is\_pal[i+1][j-1]);  
 arr[i][j] = arr[i][j-1]+arr[i+1][j]-arr[i+1][j-1]+is\_pal[i][j];  
 }  
 }  
 while (n--) {  
 int l, r;  
 scanf("%d%d", &l, &r);  
 printf("%d\n", arr[--l][--r]);  
 }  
}



**Problem 1472C**

Let's look at some starting position j. After making a move from it, we will get a[j] points and move to the position j+a[j], continuing the same game. This means that by choosing the position j, we can assume that we will get a result a[j] more than if we chose the position j+a[j]. Formally, score(j)=score(j+a[j])+a[j].

Let's calculate all the results of the score and store them in an array. Let's start iterating through the positions from the end, then being in position i we will know score(j) for all j>i. Using the formula above, we can calculate score(i) in one operation. It remains only to choose the maximum of all such values.

#include <stdio.h>  
int main(){  
 int i,n,t;  
 scanf("%d",&t);  
 while(t--){  
 scanf("%d",&n);  
 long long int a[n], max=0;  
 for(i=0;i<n;i++){  
 scanf("%lld",&a[i]);  
 }  
 for(i=n-1;i>=0;i--){  
 if(i+a[i]<n){  
 a[i]+=a[i+a[i]];  
 }  
 if(a[i]>max){  
 max=a[i];  
 }  
 }  
 printf("%d\n",max);  
 }  
 return 0;  
}

